Monte Carlo Tree Search for Simultaneous Move Games: A Case Study in the Game of Tron

Marc Lanctot Christopher Wittlinger Mark H.M. Winands
Niek G.P. Den Teuling

Department of Knowledge Engineering, Maastricht University,
P.O. Box 616, 6200 MD Maastricht, The Netherlands

Abstract

MCTS has been successfully applied to many sequential games. This paper investigates Monte Carlo Tree Search (MCTS) for the simultaneous move game Tron. In this paper we describe two different ways to model the simultaneous move game, as a standard sequential game and as a stacked matrix game. Several variants are presented to adapt MCTS to simultaneous move games, such as Sequential UCT, Decoupled UCT, Exp3, and a novel stochastic method based on Regret Matching. Through the experiments in the game of Tron on four different boards, it is shown that Decoupled UCB1-Tuned perform best, winning 62.3% of games overall. We also show that Regret Matching wins 53.1% of games overall and search techniques that model the game sequentially win 51.4-54.3% of games overall.

1 Introduction

Games are important domains for investigating intelligent search techniques. Classic examples include Chess, Checkers and Go, each of which are simple to learn yet hard to master. In classic game tree search, game-specific knowledge is used to determine the strength of each position using a static evaluation function. If the evaluation function is difficult to design then an alternative approach has to be chosen.

Monte Carlo Tree Search (MCTS) [4, 6, 10] builds up a search tree without requiring an evaluation function. Instead, it builds a search tree gradually guided by Monte Carlo simulations. MCTS was initially applied to the game of Go [6] but has since been applied to many different games and settings [3]. This paper focuses on selection and backpropagation strategies in MCTS applied to two-player turn-based simultaneous move games, such as Tron. Algorithms investigated in this paper, including sequential UCT [10], are: UCB1-Tuned, Decoupled UCT, Decoupled UCB1-Tuned, Exp3 and Regret Matching.

In Tron, two players move simultaneously through a discrete grid and at each move create a wall behind them. The first applications of MCTS to Tron [7, 15] applied standard (sequential) UCT while treating the game as a turn-based alternating move game in the search tree. A comparison of selection and backpropagation strategies in simultaneous move MCTS is presented in [13]. However, results are only presented for a single map and there is no comparison to sequential UCT previously used in this domain. In addition, we introduce a new variant based on Regret Matching, which performs relatively well in practice. Throughout this paper, we investigate the impact of different selection and backpropagation strategies on the playing performance of MCTS in Tron.

The paper is organized as follows. It starts with a brief description of Tron and MCTS in Section 2. Section 3 deals with how MCTS handles the game specific principles of Tron. In Section 4 the different selection strategies are explained. Afterwards experiments are shown in Section 5 and a conclusion is drawn from the experiments in Section 6. Furthermore, possible future research is also discussed in Section 6.

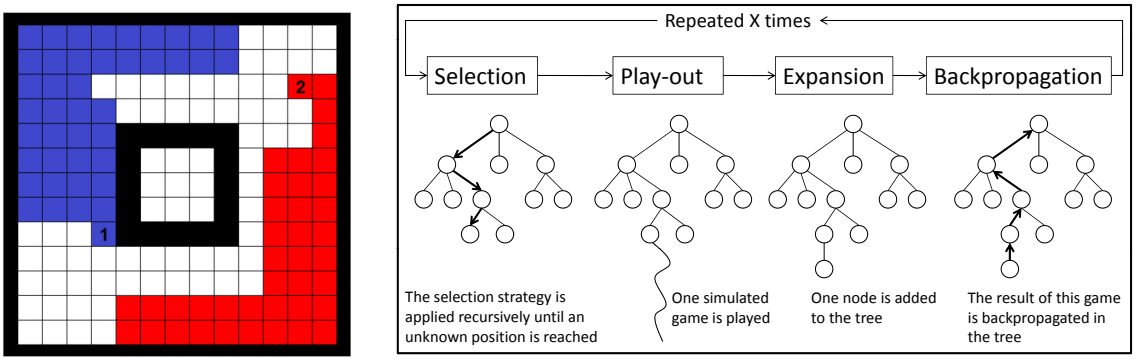


Figure 1: (Left) A game in Tron; 41 moves are already played. Player 1 started in the top left corner and Player 2 started in the bottom right corner. (Right) The four phases of the Monte Carlo Tree Search.

2 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) [6, 10] is a technique used for decision-making in turn-based, sequential games. To make a decision, MCTS makes use of simulations combined with an incrementally-built search tree. In the search tree, each node represents a state in the game. To evaluate a state, a game is simulated in self-play from the current (root) state of the game until the game is finished. The first part of each simulation will encounter states that are part of the search tree. Which states are encountered in this first part depends on the moves selected during the simulation (the so-called *selection strategy*). When a state is not in the tree, a *play-out policy* chooses successor states until the end of the game. MCTS then *expands* the tree by adding the first state it encountered along its play-out. The result of the simulation is then *backpropagated* to every node visited in the simulation up to the root where node statistics are updated accordingly. The right part of Figure 1 illustrates the four different phases of a simulation [5].

2.1 Simultaneous Move MCTS

Standard MCTS applies to sequential turned-based games. However, in some games moves are chosen simultaneously. There are two models to handle the simultaneous moves that occur in Tron.

The first model, used to implement Sequential UCT, ignores the presence of simultaneous moves and treats the game as a sequential turn-based game inside the search tree. We call this the *sequential tree model*. In practice, this worked well in Tron [7, 15], except it clearly favours one player which is especially problematic when players are close to each other. In this model, the game is sequential inside the search tree, until a leaf node is reached. The play-outs are then simulated as a simultaneous game [7]. In this paper, the player running the search always moves first.

The second model, used to implement simultaneous move MCTS, stores a matrix at each node. We call this the *stacked matrix model*. Each cell of the matrix corresponds to a joint move, i.e., a move chosen by both players simultaneously, and a corresponding child node (successor state). This is a more accurate representation of the underlying game since players are not able to influence their current decision after having seen the other player's current move choice. This is the model used in [11] and [13].

3 Tron & MCTS

Tron originates from the 1982 movie with the same name. It is a two-player game (see left part of Figure 1) played on discrete grids possibly obstructed by walls. In addition, the maps are mostly symmetric so that none of the players have an advantage. Unlike sequential turn-taking games where payers play consecutively,

at each step in Tron both players move simultaneously. The game is won if the opponent crashes into a wall or moves off the board. If both players crash at the same turn into a wall, the game ends in a draw.

Tron is played in a grid-like environment and often the two players become separated from each other. When this happens, each agent is essentially playing their own single-player game and the goal of the game becomes to outlast the opponent. Therefore the play-out can be prematurely terminated by counting the number of squares captured by each player and then assigning a win to whoever has claimed the most space. The problem is that some positions might not offer a way back and therefore become suicide moves. For that reason a greedy wall-following algorithm can be used, which tries to fill out the remaining space. When both players have filled their space, the moves which were made are counted and the player with the higher move count wins. This approach was proposed by Den Teuling [7].

Also, when players are separated, there is no need to let a play-out decide which player would win. Instead, it can be predicted by the Predictive Expansion Strategy (PES) [7]. PES is used to avoid play-outs when they are not necessary. Each time the non-root player tries to expand a node, the PES checks whether the two players are separated from each other. If this is the case, space estimation is used to predict which player would win. Finally, the expanded node becomes a leaf node and no more play-outs have to be done when reaching this node again.

4 Selection and Backpropagation Strategies

In the following subsections, different selection and update strategies for MCTS are introduced including deterministic strategies such as Sequential UCT and UCB1-Tuned, Decoupled UCT and Decoupled UCB1-Tuned, as well as stochastic strategies, which include Exp3 and Regret Matching.

4.1 Sequential UCT

The most common selection strategy is the Upper Confidence Bounds for Trees (UCT) [10]. The UCT strategy uses the Upper Confidence Bound (UCB1 [12]) algorithm. After each child has been at least selected once, UCB1 is used to select a child. This algorithm maintains a good balance between exploration and exploitation. UCB1 selects a child node k from a set of nodes K, from parent node j by using Equation 1:

$$k = \underset{i \in K}{\operatorname{argmax}} \left\{ \bar{X}_i + C\sqrt{\frac{\ln(n_j)}{n_i}} \right\}, \tag{1}$$

where n_i is the number of visits of child node i and $\bar{X}_i$ is the sample mean of the rewards of child node i. The parameter C is usually tuned to increase performance. Sequential UCT was first applied to Tron in [15].

An enhancement to the UCT selection strategy can be made by replacing the parameter C by an upper bound of the variance of the rewards [13]. This is either $\frac{1}{4}$, which is an upper bound of the variance of a *Bernoulli* random variable, or an upper confidence bound computed using Equation 2 which have the parameters the parent node j and some child node i. This variant is referred to as UCB1-Tuned [12]. Then, a child node k is selected from parent node j:

$$k = \underset{i \in K}{\operatorname{argmax}} \left\{ \bar{X}_i + \sqrt{\frac{\min(\frac{1}{4}, \operatorname{Var}_{UCB1}(j,i)) \ln(n_j)}{n_i}} \right\}, \quad \operatorname{Var}_{UCB1}(j,i) = \bar{s}_k^2 + \sqrt{\frac{2\ln(n_j)}{n_i}}, \tag{2}$$

where $\bar{s}_k^2$ is the sample variance of the observed rewards for child node k.

4.2 Decoupled UCT

Unlike standard sequential UCT and UCB1-Tuned used in the sequential tree model, Decoupled UCT (DUCT) applies UCB1 selection in the stacked matrix model for each player separately. In DUCT, a node

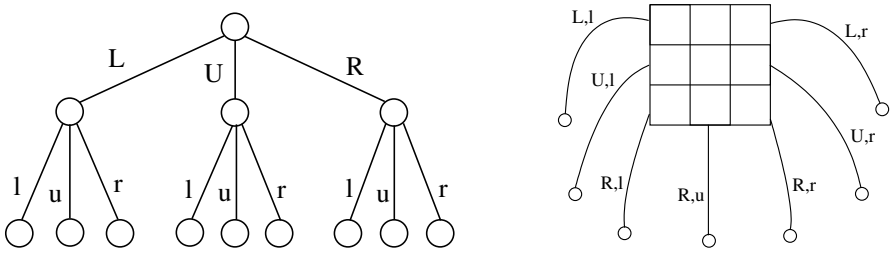


Figure 2: (Left) The sequential tree model. Each node represents the child node of the corresponding move. The first level represents Player 1 moves (L, U, and R) and the second level represents Player 2 moves (l, u, r). (Right) The stacked matrix model. Each joint move leads to a child node from a combination of moves for Player 1 and Player 2 (only 7 of the 9 joint moves are shown).

stores both moves, the one from Player 1 and from Player 2. Two moves are selected, one for each player, using separate instances of UCB, each independent of the other player's choice. To better illustrate the difference between these two concepts, see Figure 2. In the left figure, the sequential tree model is shown. In the right figure, each node contains two moves, one belonging to Player 1 and one to Player 2. These combinations are called *joint moves*. Because each level in the search tree represents now one step in the game, the branching factor is nine instead of three.

When selecting a child node, DUCT applies the default UCB1 algorithm which was described in Equation 1 with the statistics from each player's perspective independently. After a move is selected for Player 1, the selection process is repeated for Player 2 (without knowledge of the choice made by Player 1) using the statistics from Player 2's perspective. These two moves are combined to form a joint move. The final move to be played, after many simulations, can be selected in two different ways. The first, DUCT(max), selects the move with the most visits. DUCT(mix) normalizes the visit counts and samples a move according to this distribution, i.e., using a mixed strategy [11]. DUCT(max) was first used in general game-playing programs [8]. DUCT(mix) was first proposed by [16] and was also applied in a small Poker game [14].

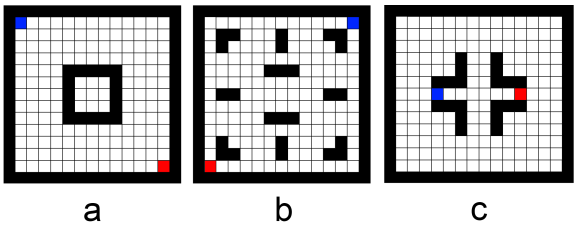
Just as an enhancement can be made by replacing the parameter C by an upper bound of the variance of the rewards in UCT, it can also be made to DUCT. Each time a node is selected and a joint move is chosen, Equation 2 is used. We refer to this variant as Decoupled UCB1-Tuned (DUCB1T).

4.3 Exp3

To this point, except for the final move selection in DUCT(mix), policies for selecting moves have been deterministic. Exp3 [1] belongs to the group of stochastic selection strategies, which means that there is a random factor involved and instead moves are sampled according to some probability distribution. Exp3, as DUCT, always uses the stacked matrix model and hence selects a joint move. Exp3 stores a list of estimated sums of rewards $\hat{X}_{a_k^p}$, where a_k^p refers to player p's move k. From the list of payoffs, a policy P is created. The probability of choosing move a_k^p of policy P is shown in Equation 3,

$$P_{a_k^p} = \frac{e^{\eta\omega(a_k^p)}}{\sum\limits_{i \in K_p} e^{\eta\omega(a_i^p)}}, \qquad \hat{X}_{a_k^p} = \hat{X}_{a_k^p} + \frac{r^p_{a_{k_1}, a_{k_2}}}{\sigma_{a_k^p}}, \tag{3}$$

where K_p is the set of moves from player p, ω can be scaled by some constant η, $r^p_{a_{k_1}, a_{k_2}}$ is the reward of the play-out when Player 1 player chose move k_1 and Player 2 chose move k_2 and is give in respect to player p. As in [11], we set $\eta = \gamma / |K_p|$. In standard Exp3, $\omega(a_k^p) = \hat{X}_{a_k^p}$, but in practice we use $\omega(a_k^p) = \hat{X}_{a_k^p} - \text{argmax}_{i \in K_p} \hat{X}_{a_i^p}$ since it is equivalent and more numerically stable [11]. The move selected is then sampled from the mixed strategy where move a_k^p is selected with probability $\sigma_{a_k^p} = (1-\gamma) P_{a_k^p} + \frac{\gamma}{|K_p|}$.



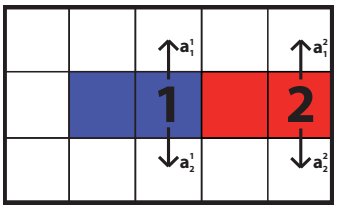


Figure 3: Left: Different boards are used for the experiments in the round-robin tournament. Right: A situation whose optimal strategies require mixing with probability distribution $(0.5, 0.5)$.

Parameter γ can be optimized by tuning it. The update of $\hat{X}_p(a_n)$ after selecting a joint move (a_1, a_2), by using the probability $\sigma_i(a_j)$, which returned some simulation result of a play-out $r^p_{a_{k_1},a_{k_2}}$ is given in Equation 3. As in DUCT(mix), the final move is sampled from the normalized visit count distribution.

4.4 Regret Matching

Regret Matching (RM) [9], as Exp3 and DUCT, always selects a joint move. Opposed to the other strategies, Regret Matching stores a matrix M with the estimated mean of the rewards (See Equation 4, where $\bar{X}_{m,n}$ is the mean of the rewards for Player 1 when the joint move $(a_1 = m, a_2 = n)$ was selected):

$$M = \begin{bmatrix} \bar{X}_{1,1} & \bar{X}_{2,1} & \bar{X}_{3,1} \\ \bar{X}_{1,2} & \bar{X}_{2,2} & \bar{X}_{3,2} \\ \bar{X}_{1,3} & \bar{X}_{2,3} & \bar{X}_{3,3} \end{bmatrix} \qquad \begin{array}{l} \text{for all } a^1_i \in K_1, R_{a^1_i} \leftarrow R_{a^1_i} + (reward^1_{i,n} - r^1_{a_m,a_n}) \\ \text{for all } a^2_i \in K_2, R_{a^2_i} \leftarrow R_{a^2_i} + (reward^2_{m,i} - r^2_{a_m,a_n}) \end{array} \quad (4)$$

Additional to matrix M, two lists are stored which keep track of the cumulative regret for not taking move a^p_k, denoted $R_{a^p_k}$. The regret is a value, which indicates how much the player regrets not having played this move. A policy P is then constructed created by normalizing over the positive cumulative regrets (e.g., if the regrets are $R_{a^1_1} = 8.0$, $R_{a^1_2} = 5.0$ and $R_{a^1_3} = -4.0$, then the policy is the probability distribution $(\frac{8}{13}, \frac{5}{13}, 0)$. As in Exp3, the selected move is sampled from $\sigma_{a^p_k} = (1-\gamma)P_{a^p_k} + \frac{\gamma}{|K_p|}$. where the variable γ can be tuned to increase performance as in Exp3.

Initially, all values in matrix M and all values in the regret lists are set to zero. After the play-out is finished and the result $(r^p_{a_m,a_n})$ for player p gets backpropagated, the cumulative reward values for each cell are updated using $X_{m,n} = X_{m,n} + r^p_{a^1_m,a^2_n}$ and the regret values are updated using the right side of Equation 4, where $reward^p_{m',n'} = r^p_{a_m,a_n}$ if $(m', n') = (m, n)$ or $\bar{X}_{m,n}$ otherwise. The final move is selected using the average of all the mixed strategies used over all simulations as described in [11].

5 Experiments

In this section the MCTS variants are evaluated. In order to make it a fair comparison between the two search tree models using a common implementation, each agent is allowed to simulate a fixed number of simulations (100,000) which took roughly one second on an AMD Opteron 6174 running at 2.2 Ghz.

The experiments are run on four different boards, three with obstacles (see Figure 3 (a), (b), and (c)) and an empty board (d), all with dimensions of 13×13. On each board 500 games are played per player matchup, with sides swapped halfway. The play-out strategy, which is used in all experiments, is the random strategy with play-out cut-offs enhancement mentioned in Section 3. However, to avoid slowing down playouts too much, they are only applied every 10 steps. The predictive expansion strategy is also used.

Before running performance experiments, some parameters (C in UCT, γ in Exp3 and Regret Matching) are tuned. As reference constants, values are used which were taken from different sources [7, 13]. Parameters C and $\gamma \in [0, 1]$ were tuned manually by playing several games against other variants. The tuned values can be seen in Table 4.

Board a	UCT	UCB1T	DUCT(max)	DUCT(mix)	DUCB1T(max)	DUCB1T(mix)	Exp3	RM
UCT	-	44%	51%	58%	40%	45%	53%	41%
UCB1T	56%	-	67%	68%	48%	55%	60%	51%
DUCT(max)	49%	33%	-	53%	28%	32%	73%	35%
DUCT(mix)	42%	32%	47%	-	28%	33%	58%	35%
DUCB1T(max)	60%	52%	72%	72%	-	59%	78%	63%
DUCB1T(mix)	55%	45%	68%	67%	41%	-	67%	48%
Exp3	47%	40%	27%	42%	22%	33%	-	33%
RM	59%	49%	65%	65%	37%	52%	67%	-
Board b	UCT	UCB1T	DUCT(max)	DUCT(mix)	DUCB1T(max)	DUCB1T(mix)	Exp3	RM
UCT	-	50%	51%	57%	49%	51%	72%	50%
UCB1T	50%	-	53%	65%	50%	52%	70%	56%
DUCT(max)	49%	47%	-	63%	41%	46%	79%	52%
DUCT(mix)	43%	35%	37%	-	24%	32%	70%	38%
DUCB1T(max)	51%	50%	59%	76%	-	55%	83%	62%
DUCB1T(mix)	49%	48%	54%	68%	45%	-	80%	54%
Exp3	28%	30%	21%	30%	17%	20%	-	23%
RM	50%	44%	48%	62%	38%	46%	77%	-
Board c	UCT	UCB1T	DUCT(max)	DUCT(mix)	DUCB1T(max)	DUCB1T(mix)	Exp3	RM
UCT	-	64%	57%	56%	46%	54%	56%	50%
UCB1T	36%	-	44%	60%	42%	47%	62%	52%
DUCT(max)	43%	56%	-	55%	42%	49%	57%	41%
DUCT(mix)	44%	40%	45%	-	28%	36%	57%	49%
DUCB1T(max)	54%	58%	58%	72%	-	60%	75%	61%
DUCB1T(mix)	46%	53%	51%	64%	40%	-	64%	55%
Exp3	44%	38%	43%	43%	25%	36%	-	36%
RM	50%	48%	59%	51%	39%	45%	64%	-
Board d	UCT	UCB1T	DUCT(max)	DUCT(mix)	DUCB1T(max)	DUCB1T(mix)	Exp3	RM
UCT	-	46%	47%	57%	44%	48%	52%	48%
UCB1T	54%	-	51%	60%	50%	52%	57%	53%
DUCT(max)	53%	49%	-	64%	35%	45%	58%	48%
DUCT(mix)	43%	40%	36%	-	29%	35%	39%	30%
DUCB1T(max)	56%	50%	65%	71%	-	55%	65%	56%
DUCB1T(mix)	52%	48%	55%	65%	45%	-	56%	50%
Exp3	48%	43%	42%	61%	35%	44%	-	42%
RM	52%	47%	52%	70%	44%	50%	58%	-

Table 1: Results of the different variants playing on boards a, b, c and d. Percentages refer to the win rate of the row player.

Total	UCT	UCB1T	DUCT(max)	DUCT(mix)	DUCB1T(max)	DUCB1T(mix)	Exp3	RM
UCT	-	50%	51%	55%	46%	49%	57%	48%
UCB1T	50%	-	53%	61%	48%	51%	60%	52%
DUCT(max)	49%	47%	-	58%	39%	45%	63%	45%
DUCT(mix)	45%	39%	42%	-	31%	36%	55%	40%
DUCB1T(max)	54%	52%	61%	69%	-	56%	70%	59%
DUCB1T(mix)	51%	49%	55%	64%	44%	-	64%	51%
Exp3	43%	40%	37%	45%	30%	36%	-	34%
RM	52%	48%	55%	60%	41%	49%	66%	-

Table 2: Overall results of the different variants playing against each other over all boards. Percentages refer to the win rate of the row player.

Variant	a	b	c	d	Total
DUCB1T(max)	65%	62%	62%	59%	$62.32 \pm 0.56\%$
DUCB1T(mix)	56%	57%	53%	53%	$54.82 \pm 0.61\%$
UCB1T	58%	57%	49%	54%	$54.32 \pm 0.55\%$
RM	56%	52%	51%	53%	$53.13 \pm 0.62\%$
UCT	47%	54%	55%	49%	$51.39 \pm 0.55\%$
DUCT(max)	43%	54%	49%	50%	$49.05 \pm 0.61\%$
DUCT(mix)	39%	40%	43%	36%	$39.51 \pm 0.64\%$
Exp3	35%	24%	38%	45%	$35.47 \pm 0.61\%$

Table 3: Results of the different variants played against each other; $\pm$ refers to 95% confidence intervals.

Parameter	Reference constant(s)	Tuned value
C	10 and 3.52 from [7, 13]	1.5
γ(Exp3)	0.36 [13]	0.3
γ(Regret Matching)	0.025 [11]	0.3

Table 4: Tuned parameter values.

6×6 Board	UCT	UCB1T	DUCT(max)	DUCT(mix)	DUCB1T(max)	DUCB1T(mix)	Exp3	RM	Total
UCT	-	43%	50%	49%	50%	48%	50%	49%	48.3 ± 0.4%
UCB1T	57%	-	50%	51%	50%	50%	51%	50%	51.2 ± 0.4%
DUCT(max)	50%	50%	-	54%	50%	52%	51%	50%	51.0 ± 0.2%
DUCT(mix)	51%	49%	46%	-	46%	47%	49%	48%	47.9 ± 0.5%
DUCB1T(max)	50%	50%	50%	54%	-	51%	51%	50%	51.0 ± 0.2%
DUCB1T(mix)	52%	50%	48%	53%	49%	-	51%	49%	50.5 ± 0.3%
Exp3	50%	49%	49%	51%	49%	49%	-	38%	48.0 ± 0.4%
RM	51%	50%	50%	52%	50%	51%	62%	-	52.2 ± 0.4%

Table 5: Results of the different variants playing against each other on a 6×6 board. Percentages refer to the win rate of the row player, and $\pm$ refers to 95% confidence intervals.

5.1 Round-Robin Tournaments

In this subsection, the performance of several players using different selection and backpropagation strategies are compared. This is done by playing matchups (of 500 games) of each player type against every other player type. Table 1 presents the results of the games on all four maps and Table 2 presents the average performance of all players. Table 3 presents the average performance of each agent over boards a-d.

From this, we see that the UCB1-Tuned variants perform best overall, with the decoupled version winning significantly (at least 7.5 percentage points) more often than its next three competitors. Given that the top three players use UCB1-Tuned, including DUCB1T(mix), it appears that a better way of performing exploration has a bigger impact on performance than the choice of the game model (sequential versus stacked matrix). The relative rank of DUCB1T(max), DUCT(max), and Exp3 presented here on board (d) are consistent with previous results on the open map [13]. Sequential UCT performed relatively well, winning 51.4-54.3% overall despite using a sequential model of the game. This could be because Sequential UCT learns to play safely since it chooses the move that leads to a situation where the advantage of the opponent's best counter-move is minimized. In fact, in the classic minimax setting the value computed in this model is a lower bound of the true optimal value [2].

As in Goofspiel [11], Regret Matching outperforms Exp3, DUCT(max), and DUCT(mix). However unlike previous results in Goofspiel, Exp3 does not outperform DUCT(max), possibly because mistakes caused by uncertainty in the final move selection are easy to recognize and exploit in Tron. Also, results of the algorithms vary from board to board, which is consistent with previous experiments in Tron [7]. Board (b), for instance, can lead to many situations where a mistake made is particular difficult to recover from. In this situation, the deterministic strategies tend to perform better since they avoid mistakes.

The round-robin tournament was repeated on a smaller (6×6) board. From the tournament on the smaller 6×6 board, Table 5, the relative performance differences are less extreme, possibly because each one is acting closer to optimally and there is less opportunity to make mistakes. The deterministic strategies decrease and the performance of the stochastic strategies increase, with Regret Matching performing best in the small board, possibly because the stochastic strategies are finding more situations where mixing is important. For example, in Figure 3 (right) both players have two possible moves. If both players choose a_1^p ("Up"), Player 1 wins and if both players choose a_2^p ("Down"), Player 1 also wins. The optimal strategy for both players is to play with a mixed strategy, where each move is chosen with probability 0.5.

6 Conclusion and Future Research

In this paper, several selection and backpropagation strategies were introduced for MCTS in Tron. The performance of these variants was studied for four different boards. Overall, the UCB1-Tuned variants perform best. Furthermore deterministic strategies appear to perform generally better than the stochastic strategies in Tron. Experiments also suggest that board layout can influence performance.

For future research, we aim to do more experiments with different boards. A hybrid selection strategy could be tested which uses a deterministic strategy if both players are far away from each other and a stochastic one as soon as both players come fairly close to each other. Also, one could try purifying the final move distributions by setting the low-probability actions to 0 and renormalizing the remaining probabilities.

Acknowledgements. This work is partially funded by MARBLE: Maastricht Research Based Learning and the Netherlands Organisation for Scientific Research (NWO) in the framework of the project Go4Nature, grant number 612.000.938.

References

[1] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. Gambling in a rigged casino: The adversarial multi-armed bandit problem. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 322–331, 1995.

[2] B. Bošanský, V. Lisý, J. Čermák, R. Vítek, and M. Pěchouček. Using double-oracle method and serialized alpha-beta search for pruning in simultaneous moves games. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 48–54, 2013.

[3] C.B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

[4] G.M.J-B. Chaslot. *Monte-Carlo Tree Search*. PhD thesis, Department of Knowledge Engineering, Maastricht University, Netherlands, 2010. Ph.D. dissertation.

[5] G.M.J-B. Chaslot, M.H.M. Winands, H.J. van den Herik, J.W.H.M. Uiterwijk, and B. Bouzy. Progressive strategies for Monte-Carlo tree search. *New Mathematics and Natural Computation*, 4(3):343–357, 2008.

[6] R. Coulom. Efficient selectivity and backup operators in Monte Carlo Tree Search. In *CG 2006*, volume 4630 of *LNCS*, pages 72–83, 2007.

[7] N.G.P. Den Teuling and M.H.M. Winands. Monte-Carlo Tree Search for the simultaneous move game Tron. In *Proceedings of Computer Games Workshop (ECAI)*, pages 126–141, 2012.

[8] H. Finnsson. Cadia-player: A general game playing agent. Master's thesis, Reykjavík University, Reykjavík, Iceland, 2007.

[9] S. Hart and A. Mas-Colell. A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, 68(5):1127–1150, 2000.

[10] L. Kocsis and C. Szepesvári. Bandit-based Monte Carlo planning. In *Proceedings of the 15th European Conference on Machine Learning (ECML)*, volume 4212 of *LNCS*, pages 282–293, 2006.

[11] M. Lanctot, V. Lisý, and M.H.M. Winands. Monte carlo tree search in simultaneous move games with applications to Goofspiel. In *Proceedings of IJCAI 2013 Workshop on Computer Games*, 2013.

[12] N. Cesa-Bianchi P. Auer and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(3):235–256, 2002.

[13] P. Perick, D. L. St-Pierre, F. Maes, and D. Ernst. Comparison of different selection strategies in Monte-Carlo Tree Search for the game of Tron. In *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*, pages 242–249, 2012.

[14] M. Ponsen, S. de Jong, and M. Lanctot. Computing approximate Nash equilibria and robust best-responses using sampling. *Journal of Artificial Intelligence Research*, 42:575–605, 2011.

[15] S. Samothrakis, D. Robles, and S. Lucas. An UCT agent for Tron: Initial investigations. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games (CIG'10)*, pages 365–371, 2010.

[16] M. Shafiei, N. R. Sturtevant, and J. Schaeffer. Comparing UCT versus CFR in simultaneous games. In *Proceedings of the IJCAI Workshop on General Game-Playing (GIGA)*, pages 75–82, 2009.